Aplicando progressive enhancement

FIavio Henrique de Souza Almeida
09/04/2021

Siempre has querido utilizar funciones modernas de HTML5 y de CSS3, pero eras impedido por tener que soportar navegadores más antiguos? ¿Tu sitio deja de funcionar con JavaScript desactivado?

En esta publicación, te mostraré formas de aplicar el concepto de progressive enhancement atacando la tríada estructura, estilo y comportamiento para ayudarte en la difícil tarea de complacer a los griegos y troyanos que llegan a tu sitio web.

Una simple analogía

Había un gran río que atravesaba dos ciudades y dos competidores en el negocio de los cruces. El primero utilizó una canoa y el segundo un jet ski. El primero, para **mejorar la experiencia de sus usuarios**, agregó un motor a la canoa.

Todo funcionó a la perfección hasta que hubo escasez de combustible. Sin energía, el jet ski dejó de funcionar y no se hizo ningún cruce. Con la canoa, todavía era posible navegar,

incluso sin el motor de la competencia, **permitir que los usuarios sigan teniendo acceso a esta función.**

¿Es el Progressive enhancement?

El concepto de progressive enhancement define que la construcción de una página parte de una **base común y garantizada para ejecutarse en los navegadores más diversos** y enseguida agregues pequeñas mejoras aunque solo funcionen en navegadores modernos.

Si alguna de estas mejoras no es compatible con el navegador, el usuario aún podrá acceder al sitio web, aunque tenga su experiencia reducida.

Este concepto no se aplica de manera uniforme en una página, y debería pensarse de forma aislada en cuanto a estructura, estilo y comportamiento. Cada punto de la tríada se comporta de manera diferente cuando se degrada, es decir, cuando no es compatible con el navegador.Una forma de pensar sobre cada punto es a través del **criterio fail-safe**.

El criterio fail-safe

El criterio *fail safe* dice que, si una característica en particular no es compatible con el navegador, esto no debería resultar en un error, incluso sin la necesidad de un tratamiento especial por parte del programador. Este criterio incluye HTML y CSS.En HTML, cuando usamos alguna tag desconocida por el navegador, no se genera ningún error, porque **la tag simplemente se ignora**.

Un ejemplo práctico de esto es el uso de la tag

de HTML5. Podemos usarla en nuestro marcado para mejorar la semántica de nuestra página, pero si el navegador no lo admite, la lista con los enlaces de navegación seguirá siendo accesible:

```
<nav>
<ul>
    <li><a href=""#"">Productos</a></li>
    <li><a href=""#"">Promociones</a></li>
    <li><a href=""#"">Contacto</a></li>
```

```
</ul>
</nav>
```

Este comportamiento ha existido desde que web es web y su beneficio de alguna manera es consistente con el progressive enhancement de manera *out of the box*, sin la intervención del programador.

El programador front-end puede utilizar tags más modernas con la certeza de que los navegadores que no las admitan las ignorarán. Como no se genera ningún error, el usuario puede acceder al recurso deseado.

Tags no admitidas se ignoran, por lo que no se pueden aplicar estilos. Esto es un problema, especialmente si necesitas diseñar una tag contenedor como

de HTML5. Todas las versiones de Internet Explorer abajo de la versión 9 sufren este problema.

Este problema se resuelve mediante el hack.html5shiv, lo que hace que las tags de HTML5 se puedan diseñar en estas versiones de IE. Este hack se carga a través de un **comentario condicional**.

Comentario Condicional

Un comentario condicional es una forma de cargar scripts y estilos condicionados a la versión de Internet Explorer.

Un punto fuerte de esta técnica es que solo IE puede entenderla, siendo ignorada por otros navegadores. En el siguiente ejemplo, se realizarán ajustes específicos para cada versión de IE:

```
<!--[if IE 6]>

<link rel="stylesheet" type="text/css" href="solo-para-ie6.css">

<![endif]-->

<!--[if IE 7]>
```

```
<link rel="stylesheet" type="text/css" href="solo-para-ie7.css">

<![endif]-->
```

Internet Explorer 10 no admitirá comentarios condicionales.

CSS y progressive enhancement

Como HTML, CSS también es *fail-safe*: si alguna propiedad no existe, será ignorada sin comprometer el acceso al recurso por parte del usuario. Además, esta característica puede proporcionar al mismo tiempo resultados aceptables.

Un ejemplo clásico de diseño progresivo:

```
h1{
    background-color: rgb(127, 214, 110);
    background-color: rgba(127, 214, 110, .1);
}
```

En el ejemplo anterior, el selector se aplica a la propiedad **background-color** un color a través de **rgb**. Esta propiedad se repite inmediatamente después, pero recibiendo el mismo color a través de **rbga** con soporte a la transparencia. El primero tiene una amplia cobertura por parte de los navegadores, mientras que el segundo, no tanto.

Resulta que, después de que se haya asignado la primera propiedad, se sobrescribirá con el valor de la segunda, solo si el navegador lo admite. Por tanto, el que carece de transparencia tendrá un fondo estanco. No ocurrirá ningún error, consagrando la naturaleza *fail-safe* de CSS.

Puedes encontrar más ejemplos en la excelente publicación css3 e progressive enhancement.

Estilos en conflicto

A diferencia de HTML, un diseño progresivo con CSS exige un poco más de atención por parte del programador front-end:

```
h1 {
    display: inline-block;
    border-style:solid;
    border-width: 1px 1px 4px 4px;
    box-shadow: -3px 3px 2px;
}
```

Para crear un efecto de sombra, se utilizó la propiedad **border-width** con soporte consistente en múltiples navegadores. Poco después, la propiedad **box-shadow del CSS3**, con menor soporte que el primero, pero con una experiencia visual mejorada.

Si el navegador no es compatible box-shadow, la propiedad border-width se aplicará, pero si la primera también es compatible, tendremos dos propiedades funcionando al mismo tiempo, lo que es un problema.

Ten en cuenta que esto no es una cuestión de *fail-safe*, aunque también puede ser parte de la ecuación, pero **una cuestión condicional**, es decir, aplicación de estilos condicionados al soporte o no de un recurso específico.

JavaScript no obstructivo y progressive enhancement**JavaScript en su naturaleza no es fail-safe**, siendo el punto más problemático de la tríada.

En lugar de aplicar el concepto *fail-safe*, la técnica de **JavaScript no obstructivo**.

La técnica JavaScript no obstructivo**, a grandes rasgos, parte de la premisa en la que los usuarios sin soporte a JavaScript podrán consumir la página, ya que el no funcionamiento de los scripts no bloqueará el acceso al contenido**.

Esta técnica cambia la forma en que se ve JavaScript, considerándolo como un **“Plus” y nunca un recurso fundamental para el funcionamiento de la página**.

Es por estas características que esta técnica es consistente con el concepto de mejora progresiva, ya que una base sólida y garantizada para funcionar en los más diversos navegadores es una base sin JavaScript. Un ejemplo:

```
window.localStorage.cep = "XXXXX-YYY";
```

El código anterior almacena el código postal ingresado por el usuario para que no tenga que ingresarlo cada vez, pero **no funcionará y generará un error** si el navegador no soporta localStorage de HTML5, como es el caso de IE 7.0 entre otros.

La ausencia de esta característica hará que la experiencia del usuario sea menos placentera (tener que teclear cada vez), pero el usuario podrá seguir comprando, por ejemplo.

Feature Detection

Una de las formas de solucionar el problema anterior es utilizar la técnica de **feature detection**, que consiste en probar la existencia de una determinada *feature* del navegador, lo que permite al programador decidir qué hacer. Entonces tenemos:

```
if (window.localStorage){
    window.localStorage.cep = “XXXXX-YYY”;
}
```

Polyfill

El programador, después de elaborar su lógica de detección, en lugar de simplemente dejar de usar el *feature* sin soporte, puede crear su propio "sustituto" o apelar a bibliotecas de terceros que mimeticen la *feature* original.

Estas bibliotecas de terceros se denominan **polyfills**. Por ejemplo, para localStorage es posible utilizar polyfill https://gist.github.com/350433.

Recuerde que incluso si usa un polyfil, debe ser un "plus", no algo sin lo que su sitio web no funcionará..

Feature Detection con Modernizr

No siempre es fácil desarrollar algoritmos de detección de recursos como en el ejemplo anterior. Además, el algoritmo de ejemplo es defectuoso, ya que la presencia de cualquier objeto con el nombre localStorage puede causar un falso positivo.

Una biblioteca que ayuda en el proceso de detección es Modernizr. Hay una serie de comprobaciones, incluso para las últimas funciones de HTML5.

Una vez detectada la ausencia de un determinado recurso, simplemente elige el polyfill (tuyo o de un tercero) de tu interés y Modernizr se encargará de cargártelo.

```
<script type="text/javascript" src="script/modernizr-custom.js"></script>

<script type="text/javascript">
    Modernizr.load({
    test: Modernizr.localstorage,
    nope: ['script/localstorage-polyfill.js']
});

</script>
```

Es posible personalizar la compilación de Modernizr con las pruebas de interés en su propio sitio web, reduciendo así el tamaño de la biblioteca final.

Aplicación condicional de estilos

Modernizr en sí ayuda a resolver el problema de nuestro CSS, cuando tenemos dos propiedades que no se pueden aplicar de forma concurrente, aplicándolas de forma condicional. ¿Cómo?

Modernizr agrega automáticamente en la tag HTML una clase para cada recurso que detecta y si el recurso no es compatible, tendrá el prefijo "no-".

El siguiente ejemplo ilustra la compatibilidad con localStorage y la ausencia de box-shadow:

```
<html class="localstorage no-boxshadow"]

<!-- restante do html -->
```

Ahora, todo lo que queda es cambiar el CSS:

```
/* aplicado solo si box-shadow es compatible */

.boxshadow h1 {
    box-shadow: -3px 3px 2px
}

/* aplicado solo si box-shadow no es compatible */
```

```
 .no-boxshadow h1 {
    border-width: 1px 1px 4px 4px;
}
```

El ejemplo anterior toma en cuenta las clases agregadas automáticamente por Modernizr, asegurando la aplicación condicional de estilos.

¿Qué pasa si JavaScript no está disponible?

Modernizr tiene un **mecanismo de fallback** limitado, pero no menos útil para situaciones en las que el soporte de JavaScript no está disponible.

El equipo de Modernizr sugiere que el programador front-end agregue la clase "no-js" a la tag . Cuando se carga Modernizr (JavaScript habilitado), cambiará automáticamente la tag "no-js" a "js". Este comportamiento permite aplicar estilos condicionales en función de la disponibilidad o no de JavaScript:

```
.boxshadow h1 {
    box-shadow: -3px 3px 2px
}

/* classe .no-js agregada */

.no-boxshadow h1, .no-js h1 {
    border-width: 1px 1px 4px 4px;
}
```

Una posible crítica a esta solución aparece cuando el navegador sin soporte JavaScript tiene soporte para la propiedad box-shadow, ya que el estilo básico se aplicará usando el truco con borde, igual habrá conformidad con el concepto de progressive enhancement, asegurando una base sólida de funcionar en los navegadores más diversos.

Conclusión

Aplicar el concepto de progressive enhancement a cada punto de la tríada estructura, estilo y comportamiento es un rompecabezas que exige aún más del programador-front.Nuevos problemas necesitan nuevas soluciones, como es el caso de la pluralidad de

dispositivos móviles, televisores e incluso refrigeradores que ahora acceden a nuestros sitios web.

Hay técnicas y recursos listos para ser utilizados, el progressive enhancement es una de ellas, todavía tenemos el responsive design entre otros.

Si te gustó este tipo de contenido les invitamos a conocer la página de Alura Latam, donde encontrarás diversos cursos de tecnología.

Puedes leer también:

- ¿Cómo funciona el import y export de JavaScript?
- Cambiando CSS con JavaScript
- ¡Organiza tu código Javascript de una manera fácil!

ARTÍCULOS DE TECNOLOGÍA > FRONT END

- Estudia las 24 horas, los 7 días de la semana
- Foro y comunidad exclusiva para resolver tus dudas
- Acceso a todo el contenido de la plataforma por 6 meses

¡QUIERO EMPEZAR A ESTUDIAR!

Paga en moneda local en los siguientes países

ANUAL

US$79,90

un solo pago de US$79,90

- 218 cursos
- Videos y actividades 100% en Español
- Certificado de participación
- Estudia las 24 horas, los 7 días de la semana

Nuevos cursos
cada semana

NAVEGACIÓN

PLANES
INSTRUCTORES
BLOG
POLÍTICA DE PRIVACIDAD
TÉRMINOS DE USO
SOBRE NOSOTROS
PREGUNTAS FRECUENTES

¡CONTÁCTANOS!

¡QUIERO ENTRAR EN CONTACTO!

BLOG

PROGRAMACIÓN
FRONT END
DATA SCIENCE
INNOVACIÓN Y GESTIÓN
DEVOPS

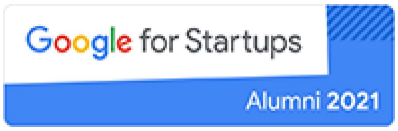
AOVS Sistemas de Informática S.A
CNPJ 05.555.382/0001-33

SÍGUENOS EN NUESTRAS REDES SOCIALES

ALIADOS

Empresa participante do SCALE UP ENDEAVOR

En Alura somos unas de las Scale-Ups seleccionadas por Endeavor, programa de aceleración de las empresas que más crecen en el país.



Fuimos unas de las 7 startups seleccionadas por Google For Startups en participar del programa Growth Academy en 2021

POWERED BY

CURSOS

Cursos de Programación
Lógica de Programación | Java

Cursos de Front End
HTML y CSS | JavaScript | React

Cursos de Data Science
Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

Cursos de DevOps
Docker | Linux

Cursos de Innovación y Gestión
Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics | Liderazgo y Gestión de Equipos | Startups y Emprendimiento